Quantum Programming in Polylogarithmic Time*

Florent Ferrari
ENS Lyon, France

Emmanuel Hainry,
Romain Péchoux,
Mário Silva
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

March 21, 2026

Abstract

Polylogarithmic time delineates a relevant notion of feasibility on several classical computational models such as Boolean circuits or parallel random access machines. As far as the quantum paradigm is concerned, this notion yields the complexity class FBQPOLYLOG of functions approximable in polylogarithmic time with a quantum random access Turing machine. We introduce a quantum programming language with first-order recursive procedures, which provides the first programming language-based characterization of FBQPOLYLOG. Each program computes a function in FBQPOLYLOG (soundness) and, conversely, each function of this complexity class is computed by a program (completeness). We also provide a compilation strategy from programs to uniform families of quantum circuits of polylogarithmic depth and polynomial size, whose set of computed functions is known as QNC, and recover the well-known separation result FBQPOLYLOG $\subsetneq$ QNC.

Introduction

Motivation. In order to check that quantum programs can be compiled and executed on a quantum computer, one has to design restrictions and constraints implying that quantum programs do not break the laws of quantum mechanics, for example, no-cloning of data and unitarity of operators. In addition, there is a need to tame their complexity in order to ensure their feasibility, that is, the fact that programs do not use too much space and time resources.

Taking inspiration from the classical world, this kind of issue has lead to the definition and study of several quantum polynomial time classes. One of the most striking examples of such classes is BQP, the quantum analog of the class of bounded-error probabilistic polynomial time problems BPP. By Yao's theorem [11], BQP corresponds exactly to what can be computed by uniform families of quantum circuits of polynomial size. This class, as well as its extension to functions, FBQP, have been characterized through various means, including function algebras [7] and first-order programs [5].

A natural question is then to study subpolynomial complexity classes. In the literature, polylogarithmic (polylog) time has been introduced and studied on Quantum Random-Access Turing Machine (QRATM) [4]. As in the classical case, this definition uses random-access machines, as opposed to standard quantum Turing machines, because of the sub-linearity of time: although the machine cannot read its entire input, it can access any input bit or qubit. On quantum models, polylog time corresponds to problems that a QRATM can solve in a polylog number of steps, leading to the definition of the complexity class FBQPOLYLOG [8, 9] of functions computable with bounded-error in quantum polylog time.

A main open problem is to design programming languages characterizing such a polylog class abstracting from the low-level considerations (machines, uniformity conditions, etc.) [10].

*This is an extended abstract of [3] presented at MFCS'25.

(Integers)	i	$\triangleq$	$\mathrm{x} \mid k \mid \mathrm{i} \pm k \mid \mathrm{i}/2 \mid \lvert\mathrm{l}\rvert$
(Booleans)	b	$\triangleq$	$\mathrm{i} \geq \mathrm{i} \mid \ldots \mid \mathrm{b} \wedge \mathrm{b} \mid \ldots$
(Qubit lists)	l	$\triangleq$	$\bar{\mathrm{q}} \mid \mathrm{l} \ominus [\mathrm{i}] \mid \mathrm{l}^{\boxminus} \mid \mathrm{l}^{\boxplus}$
(Qubits)	q	$\triangleq$	l[i]
(Statements)	S	$\triangleq$	**skip**; $\mid$ q *= $\mathrm{U}^g(\mathrm{i})$; $\mid$ S S $\mid$ **if** b **then** {S} **else** {S} $\mid$ **qcase** q **of** $\{0 \to \mathrm{S}, 1 \to \mathrm{S}\}$ $\mid$ **call** $\mathrm{proc}(\mathrm{l}_1, \ldots, \mathrm{l}_n)$;
(Procedure decl.)	D	$\triangleq$	$\varepsilon \mid$ **decl** $\mathrm{proc}(\bar{\mathrm{q}}_1, \ldots, \bar{\mathrm{q}}_n)\{\mathrm{S}\}$, D
(Programs)	P	$\triangleq$	D :: S

Figure 1: Syntax of programs

Contributions. This paper makes a first step towards solving this problem. Towards that end, we introduce a quantum programming language with first-order recursive procedures, named PLP for PolyLog Programs (Figure 1), on which we obtain the following results:

- PLP programs are terminating and reversible.
- PLP is sound for FBQPOLYLOG, i.e., each PLP program computes a function in FBQPOLYLOG. Soundness relies on the use of a bounded recursion scheme for procedures to enforce the required polylog time properties, as illustrated by the binary search example.
- PLP is complete for FBQPOLYLOG, i.e., each function of this complexity class is computed by a PLP program. Completeness is shown by a direct encoding of polylog QRATM in PLP.
- PLP is also sound but not complete for QNC. For soundness, we outline a compilation algorithm that from a PLP program and its input size outputs a quantum circuit of polylog depth and polynomial size, i.e., circuits computing functions in QNC. Completeness does not hold as it is well-known that FBQPOLYLOG is strictly included in QNC.

Related Work. A characterization of BQPOLYLOG based on a function algebra has been provided in [8, 9], where a *fast quantum recursion scheme* is used to ensure that programs terminate in polylogarithmic time. Our work employs a similar bounded recursion scheme, using a simple divide-and-conquer strategy on qubits. This characterization can be seen as simpler and more natural approach since an imperative first-order programming language is more accessible to the typical programmer.

First-Order Quantum Programs

Syntax. The considered language is a quantum programming language with first-order recursive procedures whose syntax is provided in Figure 1. There are four basic types τ for expressions: *Integer* expressions are variables x, constant $k \in \mathbb{N}$, arithmetic operations like $\mathrm{i} \pm k$ or $\mathrm{i}/2$,[1] as well as the size $\lvert\mathrm{l}\rvert$ of a list of qubits l. *Boolean* expressions are defined in a standard way. *Qubit lists* are lists of unique (i.e., non-duplicable) qubit pointers. A qubit list expression l is either a variable $\bar{\mathrm{q}}$, the first (respectively second) half $\mathrm{l}^{\boxminus}$ (resp. $\mathrm{l}^{\boxplus}$) of the qubit list l, or a list $\mathrm{l} \ominus [\mathrm{i}]$ where the i-th element of l has been removed. *Qubit* expressions are of the shape l[i], which denotes the i-th qubit in l. We also define syntactic sugar for pointing to the n-th *last* qubit in a list, by defining for any $n \geq 1$, $\bar{\mathrm{q}}[-n] \triangleq \bar{\mathrm{q}}[\lvert\bar{\mathrm{q}}\rvert - n + 1]$.

A program $\mathrm{P} \triangleq \mathrm{D} :: \mathrm{S}$ is defined in Figure 1 as a list of (possibly recursive) procedure declarations D, followed by a program statement S. Statements include the no-op instruction, (single-qubit) unitary application, sequences, conditional, quantum case, and procedure calls. For sake of universality [2], in a unitary application q *= $\mathrm{U}^g(\mathrm{i})$;, the unitary transformation $\mathrm{U}^g(\mathrm{i}) \in \mathbb{K}^{2\times 2}$ can take an

[1] The semantics of /2 will be defined as the ceiling of the result, hence it preserves the set of integers.

integer i and a function $g \in \mathbb{Z} \to [0, 2\pi)$ as optional arguments[2], and we omit them when they are of no use.

The quantum conditional **qcase** q **of** $\{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}$ allows branching by executing statements S_0 and S_1 in superposition according to the state of qubit q. When we want to treat cases on multiple qubits, we simply the nested qcases by writing **qcase** $\mathrm{q}_1, \mathrm{q}_2$ **of** $\{00 \to \mathrm{S}_{00}, 01 \to \mathrm{S}_{01}, 10 \to \mathrm{S}_{10}, 11 \to \mathrm{S}_{11}\}$.

Semantics. Let $\mathbb{B} \triangleq \{0, 1\}$ denote the set of Booleans and $\mathcal{L}(\mathbb{N})$ denote the set of lists of natural numbers, [] being the empty list. We interpret basic types as follows:

$$[\![\text{Integers}]\!] \triangleq \mathbb{Z} \qquad [\![\text{Booleans}]\!] \triangleq \mathbb{B} \qquad [\![\text{Qubit lists}]\!] \triangleq \mathcal{L}(\mathbb{N}) \qquad [\![\text{Qubits}]\!] \triangleq \mathbb{N}$$

Qubits are interpreted as integers (pointers) and qubit lists are interpreted as lists of pointers.

Given a program P, let the *length* of P be a function mapping each qubit variable $\bar{\mathrm{q}} \in Var(\mathrm{P})$ to an integer $\mathtt{len}(\bar{\mathrm{q}}) \in \mathbb{N}$. We write $\mathtt{len}_\mathrm{P}$ as a shorthand for $\sum_{\bar{\mathrm{q}} \in Var(\mathrm{P})} \mathtt{len}(\bar{\mathrm{q}})$ and $\mathtt{len}_\mathrm{P}^{<\bar{\mathrm{q}}}$ as a shorthand for $\sum_{\bar{\mathrm{q}}' \in Var(\mathrm{P}),\ \bar{\mathrm{q}}' < \bar{\mathrm{q}}} \mathtt{len}(\bar{\mathrm{q}}')$.

A *configuration* c of program P over $\mathtt{len}_\mathrm{P}$ qubits is of the shape

$$(\mathrm{S}, |\psi\rangle, A, f) \in (\text{Statements} \cup \{\top, \bot\}) \times \mathcal{H}_{2^{\mathtt{len}_\mathrm{P}}} \times (Var(\mathrm{P}) \to \mathcal{P}(\mathbb{N})) \times (Var(\mathrm{P}) \to \mathcal{L}(\mathbb{N})),$$

where $\top$ and $\bot$ are two special symbols denoting termination and error, respectively, where $|\psi\rangle \in \mathcal{H}_{2^{\mathtt{len}_\mathrm{P}}}$ is a quantum state, and where, for each qubit list $\bar{\mathrm{q}} \in Var(\mathrm{P})$, $A(\bar{\mathrm{q}})$ is the set of qubit pointers accessible from $\bar{\mathrm{q}}$ and $f(\bar{\mathrm{q}})$ is the list of qubit pointers assigned to $\bar{\mathrm{q}}$.

Given a program $\mathrm{P} \triangleq \mathrm{D} :: \mathrm{S}$, with $n = \mathtt{len}_\mathrm{P}$, let Conf_n be the set of configurations over n qubits. The initial configuration in Conf_n on input state $|\psi\rangle \in \mathcal{H}_{2^n}$ is $c_{init}(|\psi\rangle) \triangleq (\mathrm{S}, |\psi\rangle, \bar{\mathrm{q}} \mapsto \{1, \ldots, \mathtt{len}(\bar{\mathrm{q}})\}, \bar{\mathrm{q}} \mapsto [1, \ldots, \mathtt{len}(\bar{\mathrm{q}})])$. A final configuration can be defined in the same way as $c_{final}(|\psi\rangle) \triangleq (\top, |\psi\rangle, \bar{\mathrm{q}} \mapsto \{1, \ldots, \mathtt{len}(\bar{\mathrm{q}})\}, \bar{\mathrm{q}} \mapsto [1, \ldots, \mathtt{len}(\bar{\mathrm{q}})])$.

Each unitary transformation U of a unitary application q `*=` $\mathrm{U}^g(\mathrm{i})$; comes with a function $[\![\mathrm{U}]\!]$ assigning a unitary matrix $[\![\mathrm{U}]\!](g)(n) \in \mathbb{K}^{2\times 2}$ to each integer n and function $g \in \mathbb{Z} \to [0, 2\pi)$. We restrict ourselves to three kinds of gates: the phase gate Ph, rotation gate $\mathrm{R_Y}$ and NOT gate NOT, whose semantics is defined as follows:

$$[\![\mathrm{Ph}]\!](g)(n) \triangleq \begin{pmatrix} 1 & 0 \\ 0 & e^{ig(n)} \end{pmatrix} \quad [\![\mathrm{R_Y}]\!](g)(n) \triangleq \begin{pmatrix} \cos(g(n)) & -\sin(g(n)) \\ \sin(g(n)) & \cos(g(n)) \end{pmatrix} \quad [\![\mathrm{NOT}]\!](\cdot)(\cdot) \triangleq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The big-step semantics $\cdot \longrightarrow \cdot$ is defined as a relation in $\bigcup_{n\in\mathbb{N}} \mathrm{Conf}_n \times \mathrm{Conf}_n$. We write $[\![\mathrm{P}]\!](|\psi\rangle) = |\psi'\rangle$, whenever $c_{init}(|\psi\rangle) \longrightarrow c_{final}(|\psi'\rangle)$ holds. If the program P terminates on all inputs (i.e., always reaches a final configuration) then $[\![\mathrm{P}]\!]$ is a total function on quantum states.

When an input state is defined by different qubit lists, we denote them in subscript. For instance, for $x, y \in \{0, 1\}^*$ and $m \in \mathbb{N}$, we have that $|x\rangle_{\bar{\mathrm{q}}_1} |y\rangle_{\bar{\mathrm{q}}_2}$ indicates state $|x\rangle$ given as input to qubit list $\bar{\mathrm{q}}_1$, state $|y\rangle$ given as input to qubit list $\bar{\mathrm{q}}_2$.

Example (Binary search)**.** *Let $x \in 0^*1^*2^*$ be a sorted string and $\hat{x}$ denote the encoding of x as a binary given by $\hat{0} \triangleq 00$, $\hat{1} \triangleq 01$, and $\hat{2} \triangleq 10$. Program* `SEARCH` *in Figure 2 computes the function $[\![\mathtt{SEARCH}]\!](|\hat{x}\rangle_{\bar{\mathrm{q}}_1} |0\rangle_{\bar{\mathrm{q}}_2}) = |\hat{x}\rangle_{\bar{\mathrm{q}}_1} |b\rangle_{\bar{\mathrm{q}}_2}$, where $b \in \{0, 1\}$ indicates whether x contains a 1 or not.*

Polylogarithmic time restrictions. We now define some restrictions on the admissible programs to guarantee that they terminate in polylogarithmic time (i.e., each procedure cannot perform more than a polylogarithmic number of recursive calls in the input size) and that their total number of sequential procedure calls (i.e., calls that are not in superposition) is bounded polylogarithmically.

Towards that end, we define a relation between procedures to account for recursion. Given a program P, and two procedures proc and proc′ appearing in P, let $\mathrm{proc} \sim_\mathrm{P} \mathrm{proc}$ denote that proc and proc′ are mutually recursive.

[2]In the case of quantum polynomial time, Adleman et al. [1] showed how the choice of amplitudes can affect the expressivity of classes such as BQP, requiring the restriction of *polynomial-time approximable complex amplitudes*. How the set of amplitudes influences the class FBQPOLYLOG remains an open question, as discussed in [8, 9], and so we abstain from the use of the entire set of complex numbers and instead use a field $\mathbb{K}$ which may refer to, for instance, polynomial-time approximable complex amplitudes.

SEARCH

```
1  decl search(q̄1, q̄2) {
2    if |q̄1| > 1 then {
3      qcase q̄1[|q̄1|/2, |q̄1|/2 + 1] of {
4        00 → call search(q̄1^⊞ ⊖ [1], q̄2); ,
5        01 → q̄2[1] *= NOT; ,
6        10 → call search(q̄1^⊟ ⊖ [−1], q̄2); ,
7        11 → skip; }
8    } else { skip; }
9  } ::
10 call search(q̄1, q̄2);
```

Figure 2: Binary search as an example of a PLP program.

Definition. *A program* P *is said to be* recursively halving*, denoted* $\mathrm{P} \in \textsc{half}$*, if for each procedure* $\mathrm{proc} \in \mathrm{P}$ *and for each procedure call* $\textbf{call } \mathrm{proc}'(\mathrm{l}_1, \ldots, \mathrm{l}_n); \in \mathrm{S}^{\mathrm{proc}}$*,*

$$\textit{if } \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}' \textit{ then there are } 1 \leq i \leq n \textit{ and } \mathrm{l} \textit{ such that } \mathrm{l}^{\boxminus} \textit{ or } \mathrm{l}^{\boxplus} \textit{ appears in } \mathrm{l}_i.$$

This restriction can be viewed as a recursion scheme, which implies a polylogarithmic time bound on programs in HALF by ensuring that in every (mutually) recursive procedure call at least one of the input qubit lists is cut in half.

Now we impose a further condition on the number of sequential (mutually) recursive procedure calls. For that purpose, we define the *width* of a program P in the following way.

Definition. *Given a program* P*, the* width *of a procedure* $\mathrm{proc} \in \mathrm{P}$ *is defined by* $\mathrm{width}_{\mathrm{P}}(\mathrm{proc}) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}^{\mathrm{proc}})$ *where* $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S})$ *is defined inductively on statements as follows. For basic instructions,* $w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{skip};) = w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{q} \mathrel{*}= \mathrm{U}^g(\mathrm{i});) \triangleq 0$*. For the composition of statements,* $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0\,\mathrm{S}_1) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0) + w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1)$*. Both classical and quantum branching admit the same width, as we have that* $w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{if } \mathrm{b} \textbf{ then } \{\mathrm{S}_1\} \textbf{ else } \{\mathrm{S}_0\})$ *and* $w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{qcase } \mathrm{q} \textbf{ of } \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\})$ *both correspond to* $\max(w_{\mathrm{P}}^{\mathrm{proc}}(S_0), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1))$*. Finally, for procedure calls,*

$$w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{call } \mathrm{proc}'(\mathrm{l}_1, \ldots, \mathrm{l}_n);) \triangleq \begin{cases} 1, & \textit{if } \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}', \\ 0, & \textit{otherwise.} \end{cases}$$

The width *of a program* $\mathrm{width}(\mathrm{P})$ *is defined by* $\mathrm{width}(\mathrm{P}) \triangleq \max_{\mathrm{proc} \in \mathrm{P}} \mathrm{width}_{\mathrm{P}}(\mathrm{proc})$*. Let* $\textsc{width}_{\leq 1}$ *be defined by* $\textsc{width}_{\leq 1} \triangleq \{\mathrm{P} \mid \mathrm{width}(\mathrm{P}) \leq 1\}$.

Definition. *The set* PLP *of* PolyLog Programs *is defined by* $\textsc{plp} \triangleq \textsc{half} \cap \textsc{width}_{\leq 1}$.

Example. *Program* SEARCH *in Figure 2 can be shown to be in* PLP*. This program only contains one procedure* search*, that is recursive (as* $\texttt{search} \sim_{\texttt{SEARCH}} \texttt{search}$ *holds). It is then easy to check that* $\texttt{SEARCH} \in \textsc{half}$ *as the recursive procedure calls are performed either on qubit lists* $\bar{\mathrm{q}}^{\boxminus}$ *or* $\bar{\mathrm{q}}^{\boxplus}$*. Furthermore, we verify that* $\texttt{SEARCH} \in \textsc{width}_{\leq 1}$ *by computing* $\mathrm{width}_{\texttt{SEARCH}}(\texttt{search})$*:*

$$\begin{aligned} w_{\texttt{SEARCH}}^{\texttt{search}}(\mathrm{S}^{\texttt{search}}) = \max(w_{\texttt{SEARCH}}^{\texttt{search}}(\textbf{qcase } \ldots), 0) &= \max(w_{\texttt{SEARCH}}^{\texttt{search}}(\textbf{call } \texttt{search}(\bar{\mathrm{q}}_1^{\boxplus} \ominus [1], \bar{\mathrm{q}}_2);), \\ &\qquad w_{\texttt{SEARCH}}^{\texttt{search}}(\textbf{call } \texttt{search}(\bar{\mathrm{q}}_1^{\boxminus} \ominus [-1], \bar{\mathrm{q}}_2;))) \\ &= \max(1, 1) = 1. \end{aligned}$$

A Characterization of Quantum Polylog Time

PLP characterizes exactly the functions in FBQPOLYLOG, the class of quantum polylog time approximable functions. That is, programs in PLP compute functions in FBQPOLYLOG (Soundness) and, reciprocally, for any function in FBQPOLYLOG, there exists a PLP program that computes it (Completeness).

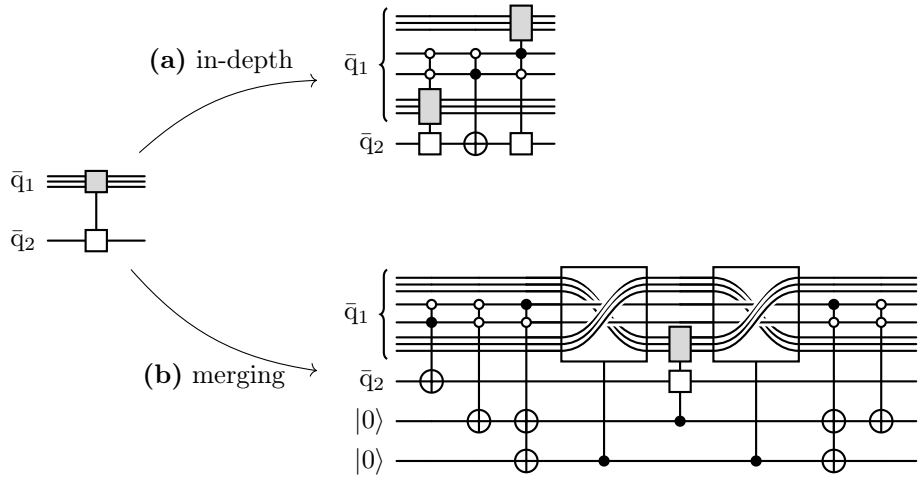

Figure 3: Compilation strategies for `search` defined in Figure 2.

Main result. We denote by $[\![\text{PLP}]\!]$ the set of functions computed by programs in PLP. That is $[\![\text{PLP}]\!] \triangleq \{[\![\text{P}]\!] \mid \text{P} \in \text{PLP}\}$. A program P approximates function $f : \{0,1\}^* \to \{0,1\}^*$ with probability $p \in [0,1]$ if $\forall x \in \{0,1\}^*, |\langle f(x)|[\![\text{P}]\!](x)\rangle|^2 \geq p$, in other words if for all input, the output of P coincides with f with probability at least p. The set of functions that can be approximated with probability at least p is denoted by $[\![\text{PLP}]\!]_{\geq p}$.

Theorem (Soundness & Completeness)**.** $[\![\text{PLP}]\!]_{\geq \frac{2}{3}} = \text{FBQPOLYLOG}$.

Proof. Soundness is proved via the fact that the HALF and $\text{WIDTH}_{\geq 1}$ restrictions imply a polylogarithmic bound for the depth of the call tree and a logarithmic bound on the degree of this tree. Then we show that if some PLP program P approximates f, then the poly-logarithmic time QRATM simulating P also approximates f and guarantees that $f \in \text{FBQPOLYLOG}$. Conversely, for completeness, given a polylog time QRATM M, we define a PLP program that simulates M. □

Circuit compilation. The compilation strategy takes inspiration from [5, 6] which in particular uses ancillas to factorize the circuits representing procedure calls in branches so as to prevent exponential blow-up of the circuit size. Their technique is called *anchoring and merging* as when a procedure call is first encountered, an ancilla is associated to this call (*anchoring*), and when a subsequent call to this procedure happens with an input of the same size, this second call is then merged with the first (*merging*). This way, instead of doubling the size of the circuit whenever recursive calls appear in separate branches of a **qcase**, as in program `SEARCH` of Figure 2, the size grows linearly in the number of nested recursive calls, hence preventing the exponential blow-up in complexity from the use of the quantum control statement [12].

Figure 3 exemplifies this phenomenon on the `SEARCH` program: the circuit on the left represented by a grey and white box the circuit for the `search` procedure applied to $\bar{q}_1, \bar{q}_2$. Since this procedure has two calls to itself, its natural compilation gives an in-depth duplication of the calls. The anchoring/merging process entails a single recursive compilation at the price of an overhead in terms of ancillary qubits and permutations. Importantly, controlled permutations can be performed in logarithmic depth, which ensures that the final circuit size and depth bounds place all PLP programs in QNC.

Theorem (Compilation)**.** *Given a* PLP *program* P*, and input size* $n = \sum_{\bar{q} \in Var(\text{P})} |\bar{q}|$*, the quantum circuit produced by the compilation process is of size* $O(n \operatorname{polylog}(n))$ *and depth* $O(\operatorname{polylog}(n))$.

References

[1] Leonard M. Adleman, Jonathan DeMarrais, and Ming-Deh A. Huang. Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540, 1997.

[2] P. Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing, 1999.

[3] Florent Ferrari, Emmanuel Hainry, Romain Péchoux, and Mário Silva. Quantum Programming in Polylogarithmic Time. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025)*, volume 345 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:17, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[5] Emmanuel Hainry, Romain Péchoux, and Mário Silva. A programming language characterizing quantum polynomial time. In *Foundations of Software Science and Computation Structures*, pages 156–175. Springer, 2023.

[6] Emmanuel Hainry, Romain Péchoux, and Mário Silva. Branch Sequentialization in Quantum Polytime. In Maribel Fernández, editor, *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*, volume 337 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[7] Tomoyuki Yamakami. A schematic definition of quantum polynomial time computability. *J. Symb. Log.*, 85(4):1546–1587, 2020.

[8] Tomoyuki Yamakami. Expressing power of elementary quantum recursion schemes for quantum logarithmic-time computability. In *Logic, Language, Information, and Computation (WoLLIC 2022)*, pages 88–104. Springer, 2022.

[9] Tomoyuki Yamakami. Elementary quantum recursion schemes that capture quantum polylogarithmic-time computability of quantum functions. *Mathematical Structures in Computer Science*, 34(7):710–745, August 2024.

[10] Tomoyuki Yamakami. Quantum first-order logics that capture logarithmic-time/space quantum computability. In Ludovic Levy Patey, Elaine Pimentel, Lorenzo Galeotti, and Florin Manea, editors, *CiE 2024*, pages 311–323. Springer, 2024.

[11] Andrew Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 352–361, 1993.

[12] Charles Yuan and Michael Carbin. Tower: Data structures in quantum superposition. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):259–288, Oct 2022.